# Второе задание по курсу ОС.

- **1.1 (fifo)** Написать программу, состоящую из двух частей (клиент и сервер) и реализующую следующий функционал. Программа-клиент получает в качестве аргумента командной строки имя файла и передаёт его содержимое серверу, используя FIFO. Сервер должен вывести полученную от клиента информацию на экран. При этом нужно обеспечить единственность клиента и сервера в системе и корректность перезапуска после убийства процесса. Из средств IPC в данной задаче разрешается использовать **только** FIFO.
  **Примечание:** программа должна состоять из двух различных файлов для клиента и сервера, т.е. процессы не имеют родственных связей. Использовать сигналы, семафоры и прочие механизмы синхронизации запрещается.
- **1.2 (signal)** Написать программу, которая получает в качестве аргумента командной строки имя файла, создаёт дочерний процесс, который читает содержимое этого файла и передаёт его содержимое родительскому процессу используя **только** сигналы.
  **Примечание:** использовать разделяемую память, очередь сообщений и другие механизмы синхронизации запрещается.
- **2.1 (msq)** Написать программу, принимающую в качестве аргумента командной строки натуральное число **n** и реализующую следующий функционал. Программа создаёт **n** дочерних процессов, которые по очереди печатают свой номер (от **1** до **n**) на экран через пробел. Для синхронизации нужно использовать очередь сообщений System V или POSIX.
- **2.2 (threads)** Написать программу, принимающую в качестве аргумента командной строки натуральное число **n** и реализующую следующий функционал. Программа создаёт **n** потоков (тредов), которые по очереди печатают свой номер (от **1** до **n**) на экран через пробел. Для синхронизации нужно использовать POSIX-семафоры.
- **3 (shmem)** Написать программу, которая передаёт содержимое файла от клиента к серверу, используя семафоры и разделяемую память (POSIX или System V). Программа должна состоять из двух файлов: клиента и сервера (т.е. процессы не имеют родственных связей). При этом нужно обеспечить единственность клиента и сервера в системе и корректность перезапуска после убийства любого из процессов.
- **4 (async)** Написать программу реализующую следующий функционал. В качестве аргументов программа принимает имя файла и натуральное число **n**. Далее родительский процесс создаёт **n** дочерних процессов, с которыми он связан при помощи **n** pipe'ов и **n** socket-пар (сокеты домена UNIX). При этом родительский процесс пишет в pipe'ы и читает из socket-пар, а каждый дочерний процесс читает из одного pipe'а и пишет в определённую socket-пару, из которой читает сервер (родительский процесс). Сервер должен работать **асинхронно**, перенаправляя информацию дальше по цепочке следующему клиенту (дочернему процессу) и поддерживая промежуточный буфер на случай недоступности того или иного файлового дескриптора. На входе первому дочернему процессу подаётся содержимое файла, которое в результате должно по цепочке передаться к последнему процессу, который выведет его на экран.